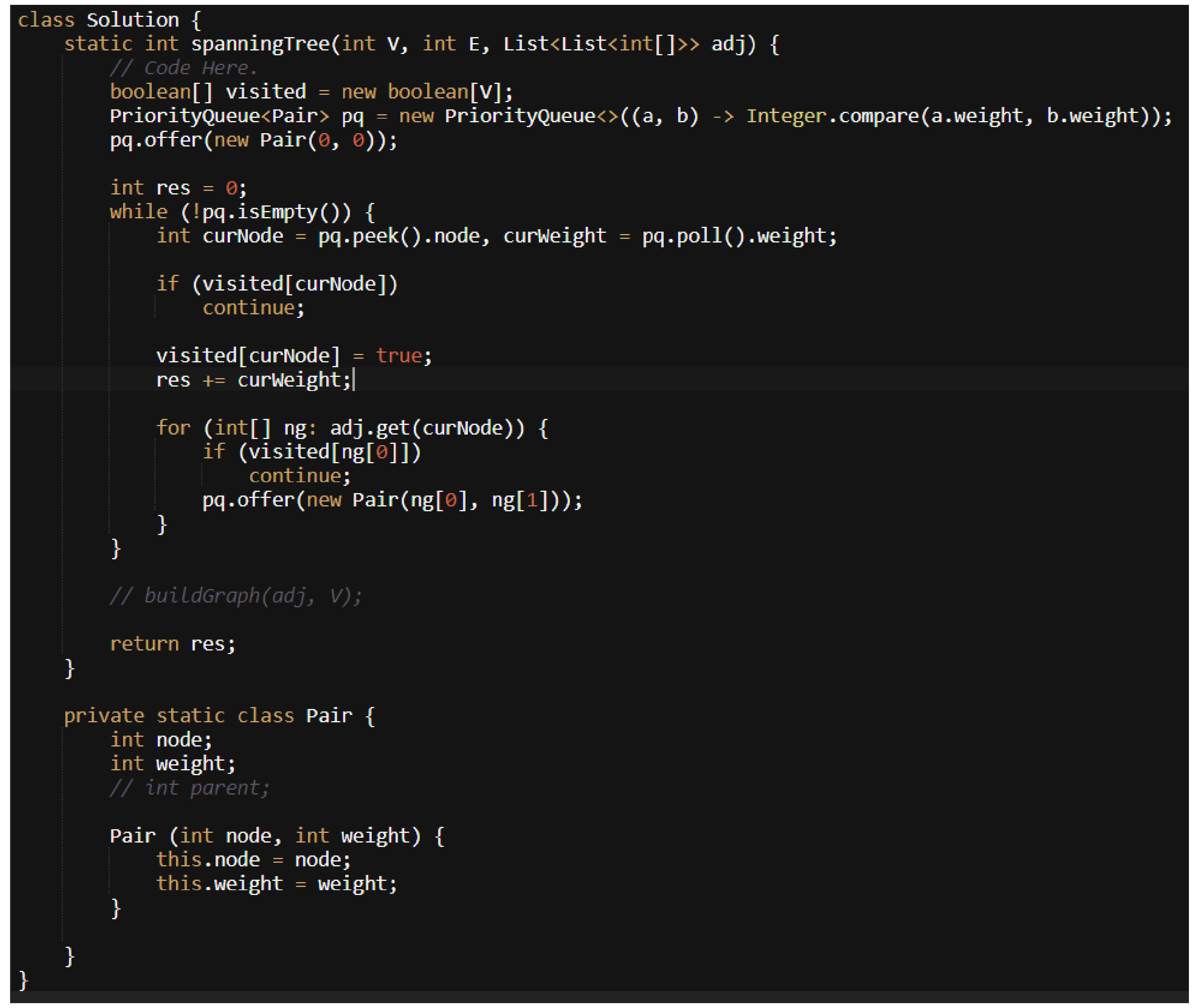
MST - Minimum Spanning Tree can be defined as a graph having 'n' nodes and 'n-1' edges. Also, all nodes shoult be reachable from each other. The sum of all the edges should be minimum. A graph can have multiple spanning trees that's why we have to choose the spannig tree with minimum sum of edges.

For PRIMS algorithm we need below data-structures.
- Priority Queue (PQ) for storing edges with their weights.
  PriorityQueue <Edge> pq = new PriorityQueue<>((a,b) -> a.weight - b.weight);
  class Edge {
      int parent;
      int node;
      int weight;
  }
- visited array/set to keep track of visited Edges.
- List of node-parent pair which keeps track of edges of our MST.
- count variable to keep track of the weight of our MST.

We follow the below steps to find MST using PRIMS algorithm.

1. We initially add in PQ new Edge(0, 0, -1) indicating zero weight to reach zero vertex & since it's the starting node, it doesn't have any parent.
2. We loop while PQ is not empty.
3. We check if current node is visited & if it is we just continue.
4. Now, we mark current node as visited, add the current weight to cost variable and if it has parent (i.e. parent is not -1) we add Pair(node, Parent) to our list of edges.
5. In this step we iterate through all the neighbors and if they are not visited we add current neighbor in PQ as Edge(parent = node, node = neighbor, weight = weight of edge node -> neighbor)
6. Once, our PQ is empty and we have the cost and list of edges which constitute our MST.

Note - For above steps we are assumming we have an adjacency list with key as our nodes and value is list of connected nodes & the weight of the edge.



```java
class Solution {
    static int spanningTree(int V, int E, List<List<int[]>> adj) {
        // Code Here.
        boolean[] visited = new boolean[V];
        PriorityQueue<Pair> pq = new PriorityQueue<>((a, b) -> Integer.compare(a.weight, b.weight));
        pq.offer(new Pair(0, 0));

        int res = 0;
        while (!pq.isEmpty()) {
            int curNode = pq.peek().node, curWeight = pq.poll().weight;

            if (visited[curNode])
                continue;

            visited[curNode] = true;
            res += curWeight;

            for (int[] ng: adj.get(curNode)) {
                if (visited[ng[0]])
                    continue;
                pq.offer(new Pair(ng[0], ng[1]));
            }
        }

        // buildGraph(adj, V);

        return res;
    }

    private static class Pair {
        int node;
        int weight;
        // int parent;

        Pair (int node, int weight) {
            this.node = node;
            this.weight = weight;
        }

    }
}
```